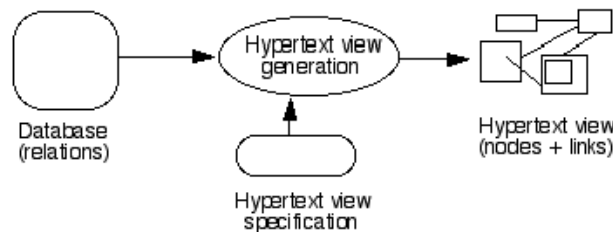


Lazy User's Manual and Reference

G. Falquet, 2003

1 Introduction

Lazy is a language and system to publish databases on the Web and to create Web application interfaces. Indeed, Lazy creates a hypertext view of a database. A hypertext view is a set of nodes (the Web pages) and hyperlinks that represent the contents of the database. In the declarative approach, the hypertext components (nodes and links) are derived from the database content (relation tuples) according to a hypertext view specification, as shown below



A hypertext view specification consists of a set of *node schemas*. Every node (Web page) is an instance of a node schema. A node schema is made of a content specification, a selection specification, and link specifications. Node schemas are written in the Lazy specification language. The Lazy system is made of a node compiler and a node server. The node compiler translates node schemas into an executable form (essentially made of SQL statements). The node server dynamically generates node instances (html or xml documents) from node schemas (in compiled form) and database elements.

2 Installation

2.1 Requirements

Since Lazy is entirely written in Java you need a **Java 1.4** run-time environment installed on your machine (you can get it from Java soft <http://java.sun.com>). Lazy does not work with Java 1.3 or earlier versions because it uses the new Java encryption scheme.

The Lazy node server is a servlet application. So you need a servlet container (server). You can either use the **tomcat** servlet container that comes with the Lazy distribution or download a more recent version from (<http://jakarta.apache.org/>).

2.2 Downloading and installing

Download the zip file from <http://cui.unige.ch/isi/lazy4/download>

1. Unzip or gunzip the file into some directory. This should create a new directory named **lazy-4.x** that contains the following subdirectories :

admin	Lazy source code of the interactive development and administration hyperspace
bin	scripts to start and manage the Lazy system
doc	documentation

examples	example hyperspaces
hsqldb	the hsql database engine (http://hsqldb.sourceforge.net/), with a demo directory that contains the data of the example Web site.
src	the Lazy source code
tomcat	(part of) the tomcat servlet container from the Apache Jakarta project (http://jakarta.apache.org/) webapps/lazy contains the lazy web application (including the lazy node server servlet)

NOTE. If you already have a tomcat server on your machine, copy **tomcat/webapps/lazy** to the **webapps** directory of your tomcat server and restart tomcat.

2.3 Starting the Lazy system with the example application

Lazy comes with an example web application for a virtual museum. It is comprised of a small database that stores information about works, artists, exhibition, etc. and a set of hypertext nodes (dynamically generated Web pages) to navigate and update the museum.

Step 1. Set environment variables

Before running the Lazy system on the example database, you must define the following environment variables:

LAZY_HOME:	the installation directory of your Lazy system (= the full path of your lazy-4.1 directory)
JAVA_HOME:	your java installation home
TOMCAT_HOME:	the home directory of your Tomcat server (set it to \$LAZY_HOME/tomcat to use the server that comes with Lazy)

Then you must run a **lazyenv** script that sets additional variables that depend on these ones.

On Unix/Linux systems with bourne shell or bash

Add the following lines to your `.login` or `.bash_profile`:

```
export LAZY_HOME=<your lazy home dir> (e.g. ~/lazy-4.1)
export $TOMCAT_HOME=$LAZY_HOME/tomcat (or the location of your Tomcat server)
export JAVA_HOME=<your java installation directory> (e.g /usr on MacOSX)
source $LAZY_HOME/bin/lazyenv.sh
```

On Unix with CShell

Add the following lines to your `.login` or `.cshrc`

```
setenv LAZY_HOME <your lazy home dir> (e.g. ~/lazy-4.1)
setenv TOMCAT_HOME $LAZY_HOME/tomcat (or the location of your Tomcat server)
setenv JAVA_HOME <your java installation directory> (e.g /usr on MacOSX)
source $LAZY_HOME/bin/lazyenv.csh
```

On Windows NT/2000/XP

Define the following variables (**Start>Settings>Control Panels>System>Advanced>Environment Variables**)

```
LAZY_HOME=<your lazy home dir> (e.g. ~/lazy-4.1)
TOMCAT_HOME=%LAZY_HOME%\tomcat (or the location of your Tomcat server)
```

JAVA_HOME=<your java installation directory> (e.g C:\jdk1.4.1)

Step 2. Start the database server

Open a new shell/terminal/command window and make sure the environment variables are correctly set. Then type

% runServer.sh (Unix/Linux/MacOSX)

C:> cd %LAZY_HOME%\bin (Windows)

C:> lazyenv

C:> runServer

Step3. Start the node server (with the provided tomcat servlet container)¹

Open a new shell/terminal/command window and make sure the environment variables are correctly set. Then type

% startns.sh (Unix/Linux/MacOSX)

C:> cd %LAZY_HOME%\bin (Windows)

C:> lazyenv

C:> startns

This starts the tomcat servlet container, which is a HTTP server that manages servlets. The node server is a servlet (named **lazy/ns**).

Step 4. Start browsing the virtual museum

With your preferred Web browser, open the example home page at **http://127.0.0.1:8080/lazy** and follow the links to the generated nodes (if your Windows machine is not connected to a network, it may take some time before it realizes that it can connect to itself, sometimes more than 1 min., you can also try to stop and restart loading).

You can also choose to explore the Lazy system by following the Lazy administration link. In this case you'll need to log in as a Lazy administrator (user = admin, password = x).

NOTE. If you already use ports 8080 and 8007 for another purpose, you can select other ports in the tomcat configuration file **\$TOMCAT_HOME/conf/server.xml**

Step 5. Stop the node server

Open a new shell/terminal/command window and make sure the environment variables are correctly set. Then type

% stopns.sh (Unix/Linux/MacOSX)

C:> cd %LAZY_HOME%\bin (Windows)

C:> lazyenv

C:> stopns

1.If you already have a tomcat server, make sure the lazy webapplication is correctly installed.

3 Using the Node Compiler

The node compiler checks the syntax of a node schema and translates it into a form that is directly executable by the node server. It can be used either from the Lazy interactive development environment or in batch mode.

The following paragraphs show how to define and compile a new node for the virtual museum example, interactively and in batch.

3.1 With the interactive development environment

The interactive development environment can be reached from the Lazy start page (<http://127.0.0.1:8080/lazy>) by following the *Lazy Administration* link. The Lazy administration page has links to projects, users, database connections, etc.

- Navigate to the *Projects* page to see all the currently defined projects.
- At this point the Lazy server may ask for a username and password, by default user **ADMIN** has password **x**.
- To define a new node in the virtual museum, follow the *Nodes* link of the *MW* project. It leads to the list of node schemas of the *MW* project.
- Click the *New Node* link to create a new node, this opens a node input page.
- In the text field type the following node schema text:

```
node myNode
  <h2>("My Artist Index") ,
  {
    <p>(name," (" , birthdate,"-", deathdate , ")")
  }
  from artist
```

- Click the *Compile* button to save and compile the node and return to the list of node schemas.
- Now there is a *myNode* line with either a red or green ball. The red ball indicates a compilation error. In this case click the pencil icon to edit the node. Once the ball is green click the *Instantiate* link to generate a Web page from this new schema.

3.2 In batch mode

- Create a new text file (say myNode.lzy) with the following content :

```
define
project MW
  node myNode
    <h2>("My Artist Index") ,
    {
      <p>(name," (" , birthdate,"-", deathdate , ")")
    }
    from artist
end
```

A source file must begin with 'define' and end with 'end', it may contain one or more node schemas. The 'project MW' statement indicates that this node belongs to the MW project.

- Open a new terminal/command window
- Make sure the environment variables are correctly set and that the lazyenv script has been executed (as indicated in the installation instructions under "Start the node server")
- Start the database server and the node server (the node compiler stores the compilation result (an executable form of the node schema) into the database).
- Compile and install your node schema by typing (Unix or Windows)

lc myNode.lzy

Once you get an error free compilation, instantiate a node by sending the following URL to the node server: **http://127.0.0.1:8080/lazy/ns?a=myNode**. It should display a list of artists together with their birthdate and deathdate.

The LAZY system is a dynamic system, once a node definition has been modified and recompiled, the new version is immediately available to the clients (there is no site generation phase). Try modifying **myNode**, recompiling it and reloading the corresponding page into your browser.

Important note. To speed up complex node generation, the Lazy node server maintains a server-side node cache. When a node schema is recompiled all its instances must be erased from the cache. However, in the current version, the batch compiler does not clear the cache. This must be done manually from the *Lazy administration* page by clicking the *Clear node cache* link.

4 Creating Hypertext Views on Existing Databases

Lazy is aimed at creating hypertext views for existing databases. There are two ways to do this, depending on where you want to keep the Lazy dictionary. The Lazy dictionary is a set of database tables that store compiled node definitions, project definitions, access rights, etc. You can either continue having HSQLDB manage the Lazy dictionary (in a the Hsqldb database that comes with Lazy) or create a fresh Lazy dictionary in the existing database.

4.1 Keeping the Lazy dictionary in the HSQL database

1. Install a JDBC driver for your database management system

If your dbms has an ODBC interface (e.g. MS Access) you can use the sun.jdbc.odbc.JdbcOdbcDriver driver that comes with the Java installation. All you have to do is to create an ODBC source connected to your database.

Otherwise you'll find the driver either in your dbms distribution or on specific web sites.

- JDBC drivers for Oracle can be downloaded from Oracle's site, they are called oracle.jdbc.driver.OracleDriver and come in .zip files (e.g. classes12.zip)
- the hsqldb driver is called org.hsqldb.jdbcDriver, it comes with the Lazy distribution in file lazy-cat/webapps/LAZY/database/hsqldb/lib/hsqldb.jar

One important point is to find the correct URL schema to access your database through JDBC. Here are some examples:

- Oracle thin client: jdbc:oracle:thin:@hostname:1521:instancename
- ODBC source: jdbc:odbc:sourcename
- hsqldb: jdbc:hsqldb:hsqldb://hostname

2. Define a new database connection and a new project

- Go to <http://127.0.0.1:8080/lazy/ns?a=ADMIN.all> and click on "Connections"
- Create a new database connection
- Define a new project and associate it to the new database connection (all the nodes of a project access the same database, given by the project's database connection).
- Start creating node schemas in the new project

That's all

4.2 Install a new Lazy dictionary in an existing database

1. Install the appropriate JDBC driver

(see above)

2. Update `Lazy.properties` and `LazyCompiler.properties`

Edit the file **\$TOMCAT_HOME/webapps/lazy/WEB-INF/classes/Lazy.properties** and modify the following lines

```
database.url=your database JDBC URL
database.user=your database user name ('sa' for hsqldb)
database.password=your database password (empty for hsqldb)
database.driver=your JDBC driver
```

Do the same in **\$TOMCAT_HOME/webapps/lazy/WEB-INF/classes/LazyCompiler.properties**. Although these files are not accessible from the Web, they should be carefully protected on your file system since they contain database passwords.

3. Modify the scripts to indicate the location of the JDBC drivers

The **JDBC_DRIVER** environment variable indicates where the JDBC drivers are. Update its definition in **\$LAZY_HOME/bin/lazyenv.xxx** by adding the location of the new driver(s).

4. Create the Lazy dictionary

Create the tables that will handle the Lazy dictionary (node definitions, users, roles, grants, etc.). The **\$LAZY_HOME/adm** directory contains the SQL scripts to create these tables

lzydict-schema.sql: the tables that stores node definitions
 or **lzydict-hsqldb-schema.sql**: the same for Hsqldb
 or **lzydict-ora-schema.sql**: the same for Oracle
admin-schema-41.sql: table schemas for all the security related tables
 or **admin-hsqldb-schema.sql** the same for Hsqldb
admin-init-41.sql: initial values for the security tables

Note. Although SQL is supposed to be the standard data definition, query, and manipulation language, every DBMS has its own interpretation of that standard. Moreover, many important functions are left undefined in the standard (data conversion, string manipulation, etc.). Hence the specific definition scripts and node definitions.

5. Recompile the Lazy administration and development nodes

Compile the node schemas that are in **\$LAZY_HOME/adm**. These nodes define the administration and development environment (to create and compile nodes, manage projects, users, access rights, connections, etc.). This is a good test to see if your **LazyCompiler.properties** file is correct. If you forgot to modify this file, the compiled nodes will be stored in the

```
% cd $LAZY_HOME/admin
% lc icon-41.lzy
Lazy node compiler 4.0b (--><< the url of your database >>)
project: ICON connectionId: DICTLAZY
node maj
node del
node new
....

% lc node-41.lzy (or node-41-for-hsqldb.lzy for Hsqldb)
...
% lc admin-41.lzy
...
```

6. Start the node server and test the installation

Unix

```
% startns.sh
```

Windows

```
C:> startns
```

From your Web browser open the URL **http://localhost:8080/lazy/ns?a=ADMIN.all** and start browsing the administration and development environment.

5 Developing with Lazy (Lazy Tutorial)

This tutorial is based on a simple virtual museum database that is made of the following tables (key attributes are in bold):

```
work (wno, author, title, c_date, height, width, picture)
artist (ano, name, birthdate, deathdate)
exhibition (exno, title, desc, organizer)
museum (mno, name, location, URL)
ownership (work, owner, acquisition)
art_cnty (artist, country, activity)
ex_content (work, exhibition, comment)
```

There is a more extensive tutorial on the isi Web site (<http://cui.unige.ch/isi/lazy/tutorial>)

5.1 Node syntax

A node scheme must be defined in a Lazy source file, which is a plain text file. It is recommended to use a suffix like .lzy or .lazy. A file that contains node definitions must begin with 'define' and end with 'end'. Nodes can also be defined directly with the interactive development environment (<http://127.0.0.1:8080/lazy/ns?a=ADMIN.all>).

A node definition takes the following form:

```
node node-name [parameter-name ...]
    element, ...
from table-name, ...
selected by expression
order by expression
```

5.1.1 Node content

The content of a node is specified by a list of elements. An *element* is either a *quoted string* (using double quotes), a *number literal*, an *attribute name*, a *parameter name*, or an *arithmetic expression*. An element or a sequence of elements can be given a markup type (an HTML or XML tag) with the following syntax:

```
<tag-name attribute=expression ...>( element , ...)
```

This notation is equivalent to the standard HTML/XML notion `<tag-name attribute=value ...> element ... </tag-name>`

Tuple elements

All the elements enclosed in { and }, are repeated for each selected tuple of the specified table(s). These elements represent the selected database tuples.

The elements placed before { and after } are generated only once. They may contain elements based on aggregate functions like sum(), max(), min(), count(). The database is queried once for computing these functions. If the selection is empty the elements outside { } are not generated, i.e. the whole node will be empty.

Example. The following node schema is intended to display a list of artists born after a given date.

```
node mw_Artist_after[d]
  <h2>( "Artists born after ", d ) ,
  {
    <p>(name," (born ",birthdate," ")
  }
  from artist
  selected by birthdate >= d
  order by name
```

An instance of this node, say **mw_Artist_after[1880]**, will be constructed as follows

1. Generate the element `<h2> Artists born after 1880</h2>`. (d is replaced by its actual value)
2. For each tuple t in artist such that t.birthdate >= 1880 generate an element of the form

```
<p>t.name (born t.birthdate )</p>
```

To see instances of this node schema, go to the IDE page, then navigate to *Projects* → *MW Nodes*. Then click the *Instantiate* link on the *mw_Artists_after* line. Type a date in the parameter field and click the *Open* button.

Alternatively, you can open the URL

```
http://127.0.0.1:8080/lazy/ns?a=MW.mw_Artist_after&a=date ,
```

where *date* is the parameter value for *d*.

5.1.2 Links

To make an element the origin of a link to another node use the following notation

href node-name[parameter-value ...](anchor-element)

Suppose that we want to add a link to the works of art of each artist in our *mw_Artist_after* node. Our example already contains a node schema *mw_Works_by_artist[id]* that displays all the works by the artist with number id. So we can simply create a link to it, as follows:

```
node mw_Artist_after_2[d]
  <h2>( "Artists born after ", d ) ,
  {<p>(href mw_Works_by_artist[ano] (name) ,
    " (born " , birthdate," ) "
  )
}
from artist
  selected by birthdate >= d
  order by name
```

About node URLs

Normally URLs are automatically generated by the Lazy system when you write an href statement. However, if you want to refer to a Lazy node from a "standard" HTML page, or directly by entering a URL in the address field of the browser, you must use the following scheme:

```
http://hostname:port/lazy/ns?a=node_name&u=parameter&u=parameter&u=...
```

(the standard port used by the tomcat server is 8080)

For example, to open the node *mw_Artist_after_2[1880]* on the local machine you must write `http://127.0.0.1:8080/lazy/ns?a=mw_Artist_after_2&u=1880`

5.1.3 Expand-in-place Links

In addition to standard hyperlinks (as found in HTML) it is possible to define expand-in-place links. These links have a different behaviour, when the anchor is clicked it is replaced, within the current page, by the referred node.

Example. Replace the **href** keyword by **expand href** in the **mw_Artist_after_2** node schema, recompile it and try clicking on links in the instance **mw_Artist_after[1880]**.

5.1.4 Inclusion links

The third kind of links is inclusion links. An inclusion link immediately include the content of the referred link at the indicated place. An inclusion link does not have any anchor text.

Replace **href mw_Works_by_artist[ano] (name)** by **include mw_Works_by_artist[ano]** in the node schema, recompile it and look at the instance **mw_Artist_after[1880]**.

Inclusion and expand-in-place links offer a very powerful mechanism to build complex nodes with heterogeneous contents. Indeed, inclusion and expand-in-place form the basic construct to jump out of the “relational box”, i.e. to build data presentation that don't look like tables or list of record.

5.2 Examples

The following node schema presents information about a work of art. It uses a one-row and two-column table. The first column (td) presents textual information and the second one displays a picture of the work. The picture's URL is stored in the database attribute picture, note how this attribute is used to give its value to the "src" attribute of the "img" element.

This node is based on two tables: work and artist because we want to display the author's name, which is not in work. So we have to join the two tables with the condition `work.author = artist.ano`

In order to show the name of the work's owner we employ another technique which consists in including the **mw_Owned** node (described below).

```
node mw_Work[id]
{
  <table>(
    <tr>(
      <td>(
        <h2>(title), <p>(c_date),
        <p>(href mw_Artist[ano] (name)),
        <p>(support," ",height," x ",width),
        <p>(include mw_Owned[wno], " ", acquired ) ,
        <p>(href mw_Works_by_artist[ano] ("Works") , " by " , name)
      ),
      <td>(
        <p>( <img width="200" src=picture>())
      )
    )
  ),
  <hr>() ,
  href mw_upd_Work[id]("Update") , " this description"
}
from work, artist
selected by work.author = artist.ano and wno = id
```

The following node is simply here to display the name of a museum whose number is given as parameter. Its purpose is to replace a numeric value (here the museum no.) by a more explicit one. This node is used by mw_Work.

```
node mw_Owned[w]
  href mw_Museum[museum](name)
  from ownership, museum
  selected by work = w and museum = mno
```

The inclusion mechanism is interesting to re-use nodes and thus to avoid writing many times the same statements. For instance, the following node defines a node footer (also called navigation bar) that can be included in many nodes

```
node mw_To_index
  <hr>() ,
  href mw_Work_index ("[Work index]") , " " ,
  href mw_Artist_index("[Artist index]") , " " ,
  href mw_Exhibition_index("[Exhibition index]")
```

This example also shows a particular use of the selection mechanism. The content of this node does not depend on any database table, it is a kind of "constant" node. However, every node must be based on a table. So we chose to base it on the naturals table and to select only one tuple from this table.

Experimenting The best way to learn the language is by doing your own experiments on the example application. Navigation to the MW project and try modifying existing nodes. You will also find all the node definitions of the museum example in the example directory of the lazy distribution.

Designing node schemas that form a good hyperspace is like designing software or buildings: there is no general method to do this. However, you can find design ideas and a design and analysis methodology in the papers published by the ISI research group [<http://cui.unige.ch/isi/reports>]

5.3 Active Nodes and Links

An active node is a node that contains active links. An active link is a reference link (href) that triggers a database action when traversed. In addition to usual elements, the source anchor of an active link can have one or more input elements and must have one action element. The general syntax of an active link is

Active href node-name[parameters] (standard-or-input-or-action-elements)

An action element takes one of the following forms:

- **on** "button-label" **do insert** table-name
- **on** "button-label" **do delete** table-name[key-attributes]
- **on** "button-label" **do update** table-name[key-attributes]

An input element takes the form:

- **set** attribute-name = expression | textfield(width) | textfield(width, initial-value) | textarea(nb-lines, width) | textarea(nb-lines, width, initial-value)

Attribute-name must refer to an attribute of the table mentioned in the action element.

Example. A node to create a new artist

```
node mw_new_Artist
<h2>("Adding a new artist") ,
active href mw_Artist_index (
  <p>("Unique identifier: " , set ano = textfield(10) , " (a number)") ,
  <p>("Name: " , set name = textfield(60) ) ,
  <p>("Birthdate: " , set birthdate = textfield(10)) ,
  <p>("Deathdate: " , set deathdate = textfield(10)) ,
  on "Add" do insert artist
)
```

Although this node will act on the "artist" table, by inserting a new tuple, its content does not depend on any particular table. When generated, this node displays the heading text, four input fields (for ano, name, birthdate, and deathdate), and an "Add" button. When the user clicks on "Add", the system takes the input values to form a new tuple, inserts this tuple into the artist table and then jumps to the node mw_Artist_index. It is not mandatory to define all the table's attributes to insert a new tuple, only those that are either part of the primary key or not "nullable" are required.

The following node enables the user to change the name of a given artist.

```
node mw_upd_name_Artist[w]
<h2>("Changing the name of artist no. " , w) ,
active href mw_Artist[w] (
  <p>("Name: " , set name = textfield(60, name) ) ,
  on "Update" do update artist[ano]
) ,
href mw_Artist[w] ("Don't change")
from artist selected by ano=w
```

This one adds 5 to the birth date of an artist (very useful indeed!).

```
node mw_add5_Artist[w]
<p>("Click on the \"Plus 5\" button to add 5 years to the birth date of " , name ) ,
active href mw_Artist[w] (
  set birthdate = birthdate * 5 ,
  on " Plus 5" do update artist[ano]
)
from artist selected by ano=w
```

Parameter values or any expression can be used as input values, as in the following node

```
node mw_add_work_exh[e, w]
<h3>("Add " , work.title , " to " , exhibition.title) ,
active href mw_Exhibition[e] (
  set work = w ,
  set exhibition = e ,
  <p>("Comment: " , set org_comment = textarea(10, 30)) ,
  on "Add" do insert ex_content
)
from exhibition, work
```

selected by exno = e and wno = w

The **set** elements must not be confused with assignment statements. The effect of a **set** takes place only after the corresponding active link has been followed (by clicking the button. Thus, elements like

```
...
set name = "Zorro",
"The name is", name
```

Will display the value of the name attribute of the currently selected tuple, not "Zorro". More on this in section 5.5.

5.4 Designing active nodes

There are basically two ways to design active nodes

1. Design "form nodes", like the mw_new_Artist node above, that let the user type data in the input fields;
2. Design navigation strategies to collect the requested information through navigation (using parameter passing to transmit data from node to node)

The following nodes show an example of the second design alternative:

```
node mw_exhibition_content[e]
{ // display the content of exhibition e
  title , . . .
}
href mw_select_work[e] ("Add a Work to this exhibition")
from ex_content, work
selected by exhibition=e and work=wno

node mw_select_work[e]
<h3>("Select a work to include in the exhibition") ,
<dl>(
  {<dd>( expand href mw_Work[wno](title) , " " ,
    href mw_add_work_exh[e, wno]("[Add]")
  )
} )
from work
```

The input values can be used as parameters to the destination node. For instance, the following node reads attribute values for a new artist, creates a tuple when the button is clicked, and then jump to a node that displays the newly created artist.

```
node mw_new_Artist_2
<h2>("Adding a new artist") ,
active href mw_Artist[ ! ano ]
<p>("Unique identifier: " , set ano = textfield(10) , " (a number)" ) ,
<p>("Name: " , set name = textfield(60) ) ,
<p>("Birthdate: " , set birthdate = textfield(10)) ,
<p>("Deathdate: " , set deathdate = textfield(10)) ,
on "Add" do insert artist
```

)

When the "Add" button is clicked, it inserts a new tuple into artist and then jump to the node mw_Artist[!ano]. The **! ano** notation indicates that the value given to the ano attribute in the **set ano = textfield(10)** element must serve as parameter to instantiate mw_Artist. In this case, it will jump to the node that displays the artist with the number we just entered.

5.5 Global variables

System variables

The Lazy node server maintains a number of "system variables" taht store information about the current session. These variables are:

[USER] : the current username

[GRP] : the datagroup the user is working in

[LANG] : the user's preferred language

They can appera anywhere in the definition of a node (in the content or in the selection part).

Example

Input variables

An input variable is a variable whose value is either entered by the user in an input field (set var = textarea, textfield, etc.) or obtained as the value of an expression (set var = expr). Variables can only be set in the definition of an active link. It is important to note that the value of an input variable is not available until the active link has been followed (by clicking the link's button). Hence it cannot be used elsewhere in the node that sets its value. However, input variable values can be passed as parameter to the target node of the active link, with the **!variable** syntax.

Within the target node, or other nodes, expressions of the form **[!variable]** yield the value of the corresponding input variable.

Examples

```
node Selector[]
  active href Works_with_name[ ! word ] (
    <p>("Word of the work's title ", set word = textfield(20)) ,
    <p>("Level of detail (1-9)", set detail = textfield(1)),
    on "Go" do display Works_with_name
  )
```

```
node Works_with_name[w]
  { title, ... , include W_Description[ ![detail]
from works
  selected by title like concat("%", concat(w, "%"))
```

Internationalized strings

To-be-written

6 Managing projects

6.1 Projects

A project is mechanism to group a set of node schema that have a common purpose. It is similar to a module in a programming language. The IDE has an environment to manage project (create new projects, delete them, add, update, and delete nodes in a project).

Every project has a set of properties that are inherited by all its nodes:

- a database connection
- formatting files (CSS, XSL, background image)
- a default node type (HTML or XML)

The full name of a node *n* belonging to project *p* is *p.n*. When referred from a node of the same project, the *p.* prefix can be omitted.

A database connection is a JDBC connection, it is comprised of a database URL, a JDBC driver (a class name), a username, and a password. Since different projects may use different connections, it is possible to build a single Lazy interface on top of several databases.

6.2 Users

A Lazy user is characterized by his or her username and password. Access rights to Lazy nodes and database tables can be assigned to users. When a new session is established, the user is automatically logged in with ther username PUBLIC. If he or she requests a node that has access restriction, he or she will be asked to enter a username an password.

In node schema, the session variable `[[USER]]` returns the username.

Note. In the current version session variables are implemented by a string substitution mechanism. If the substring `[[variable]]` appears in a string it is replaced by the variable's value. As a consequence, the string `'[[` should never appear in this form in a string.

6.3 Datagroups

Datagroups are intended to horizontally partition data between different groups of users. For instance, the same database could be used to manage data belonging to different departments. Datagroups are useful when different groups share the same data structures but not the same data. The global variable `[[GRP]]` represent the datagroup of the current session.

The dictionary schema given in appendix shows the relationships between all the security and project management related concepts. It is a UML view of the Lazy dictionary tables.

7 Encryption in Lazy

Security in Lazy

Because the property files `Lazy.properties` and `Compiler.properties` contain the database password of the db where the Lazy dictionary (if not the data) is stored, it is of the highest importance to protect them against unauthorized access. Normally, read access to these files should be granted only to the Lazy owner.

Why encrypt ?

Lazy can connect to several databases, the connection password of these databases should be protected. In addition, the user passwords are stored in the 'dictlazy' database. These password should also be protected. Finally, an probably most importantly, when Lazy is used to update a database, important infor-

mation (such as table names, attribute names, key attribute values, etc.) may appear (as hidden fields) in the pages sent to the user. A malicious user 1) get information about the database schema of the application and 2) forge pages containing updates (in forms) and send them to the server in order to slip wrong data into the database.

What is encrypted ?

When encryption is turned on, the following items are encrypted

- All user passwords
- All database connection password (except the initial 'lazydict' connection)
- All the table names, attribute names, key attribute values appearing in forms in HTML pages send to the user.

How to migrate to the encrypted form ?

The first step consists in choosing a permanent encryption key -- this key will encrypt user passwords and database connection passwords -- it must never change. This key (made of 16 hexadecimal digits) is stored in `Lazy.properties` as the `encrypt.key` attribute value.

```
encrypt.key=<16 hex digits key>
encrypt.off=on in Lazy.properties
```

Now there is a bootstrap problem: if you turn encryption on, by setting the `encrypt.off` parameter to on in `Lazy.properties`, you won't be able to log-in any more because the system will try to decrypt the stored passwords before comparing them to the typed passwords. So you'll need to manually encrypt the ADMIN password in the **secure_user** database table before proceeding to the next steps. This can be done by as follows

```
% $LAZY_HOME/bin/gpwd your-admin-password
33833A88E8F10377 <--- your password encrypted with the Lazy key
```

In the database :

```
update secure_users set pwd = '33833A88E8F10377' where userid = 'ADMIN';
```

Now you can log-in as ADMIN and set the other user's passwords (they will be encrypted)

8 Java Nodes

When complex computations are needed (or simply functions that do not exist in SQL), it is better to express them in an algorithmic language rather than hack "computation-oriented" Lazy nodes. For this purpose one can define "external nodes", the content of which are computed by calling a Java method.

The name of an external node starts with two underscore characters followed by a class name and a method name. The instantiation of such a node will call the corresponding method to generate a node content. For instance, a content element of the form **include _MW.artistAbbreviatedName[a]** will call the static method **artistAbbreviatedName** of class **MW** and include the resulting string in the node content.

A "node" method must have one of the following signatures

```
¥ static String methodName(String parameter1, ... String parameterN)
```

¥ static String *methodname*(String[] *parameter*)

```
class MW {
    static String artistAbbreviatedName(String n) {
        String af = n;
        int x = n.indexOf(" ");
        if (x > -1) af = n.charAt(0) + "." + n.substring(x+1);
        return af;
    }
    /* other methods */
}
```

. The second form is intended to implement nodes with a varying number of parameters.

For instance,

```
static String concatIf(String [] params) {
    String r = "";
    for (int i=0; i<params.length-1; i+=2) {
        if (params[i].equals("yes")) r += r.params[i+1]
    }
    return r;
}
```

This class must be compiled and the .class file stored in the webapps/lazy/WEB-INF/classes directory. Use for instance the following compilation parameters:

```
javac MW.java -d $TOMCAT_HOME/webapps/lazy/WEB-INF/classes
```

Accessing the database from an external method

The following Lazy server methods can be invoked from an external method to execute SQL statements on a database corresponding to a database connection.

```
class DBServices

/*
 * sql : the statement
 * db : name of the database connection
 * select : must be true for a 'SELECT' statement and false for
 * an 'INSERT', 'DELETE', 'UPDATE'
 */
public static QueryResult execSQLonDB(String sql,String db, boolean select)

/*
 * executes the sql statement on the LAZYDICT connection
 * (the connection to the Lazy dictionary database)
 */
public static QueryResult execSQL(String sql, boolean select)

/*
 * Query results are stored in a QueryResult object defined as follows
 */

class QueryResult {
```

```

Statement stmt;
public ResultSet result;
public boolean valid = true; // false => error in query processing
public String msg = ""; // error message if valid==false
public String sql;
public int nbUpdated = 0;

```

Examples:

```

static String updel(String keyval, String val) {
    QueryResult q1 = execSQLonDB(
        "delete from T1 where TK='"+keyval+"'", "Z_CONNECTION", false);
    QueryResult q2 = execSQLonDB(
        "update T2 set A='"+val+"' where T2K='"+keyval+"'",
        "Z_CONNECTION", false);
    String r="";
    if (!q1.valid) r += q1.msg;
    if (!q2.valid) r += q2.msg;
    return r
}

```

The **DBServices** and **QueryResult** classes are located in `webapps/lazy/WEB-INF/classes`. To compile methods that refers to them, it is necessary to add their location to the **CLASSPATH**. For instance

```

javac MW.java -classpath .:$TOMCAT_HOME/webapps/lazy/WEB-INF/classes \
-d $TOMCAT_HOME/webapps/lazy/WEB-INF/classes

```

9 Commented Lazy 4 Syntax

```

startrule = "define"
    "project" project_identifier [ "[ nodetype = ("xml" | "html" | "purexml" | "purehtml") "]" ]
    { node_def }
"end".

node_def = "node" node_identifier parameters
    [ "cachesize" = number ]
    content
    from_part

```

A node schema has a name and a (possibly empty) list of parameters. The from, select and order parts determine the list of database tuples to represent in an instance of this node. The fields define the representation of each tuple and the links to other nodes.

```
parameters = [ "[" parameter { "," parameter } "]" ]
```

```
parameter = identifier
```

```
identifier = letter { letter_or_digit | "_" }
```

content = field { "," field }

```
field = ("href" | "expand") link "(" field { "," field } ")"
      | "include" link
      | "active" "href" active_link "(" field_list ")"
      | set_attribute
      | on_action
      | content
```

A field is either the source anchor of a (possibly active) reference link, or the result of including some other node instance at that place, or a content element (and XML or HTML element with embedded elements). The fields that appear in the source anchor of an active link can set attribute values and define action to undertake when the link is activated.

field_list = field { "," field }

```
link = ( "include" | "expand""href" | "href" [ "in" target_identifier ] )
      [project_identifier "."] node_identifier
      [ "[" simple_expression | inclusion_link { "," simple_expression | inclusion_link } "]" ]
```

A link is either an inclusion (include), a reference (href), or an "expand in place" (expand href). It refers to a target node through its identifier and a list of actual parameter values (expressions). The parameter values can also be obtained as the result of an inclusion.

```
active_link = "href" [project_identifier "."] node_identifier
              [ "[" simple_expression { "," simple_expression } "]" ]
```

```
set_attribute = "set" (attribute_identifier | "parameter" | "parameter_encoding"
                      | function_identifier "(" identifier ")")
               "=" (input_field | simple_expression)
```

Defines the value that will be affected to a tuple attribute by the next "insert" or "update" action. The **set parameter** form is used to pass a value (an expression or an input value) as parameter to the target node. In this case, the **active href node** part must not contain parameter values. Parameters are positional, they must be given in the order defined in the target node schema.

```
input_field = "textfield" "(" simple_expr [ "," simple_expr ] ")"
              | "textarea" "(" simple_expr "," simple_expr [ "," simple_expr ] ")"
              | "free" "(" field ")"
              | "select" "(" ("include" link | simple_expr { "," simple_expr } ) ")"
```

A "free" input field indicates that the given field completely specifies the input method with html or xml tags. The input method must provide a value for a parameter named "av".

```
on_action = "on" string_expression "do" (
          "display"
          | "insert" table_identifier
          | ("delete" | "update") table_identifier "[" keyvalue { "," keyvalue } "]"
        )
```

Generate a clickable button with the given label and defines the associated database action. Possible actions are

- **insert t**

inserts a new tuple in table *t*, the attribute values are given by the **set attr = ...** fields that appear in the active link anchor.

- **delete t[a1 = v1, ...]**

deletes all the tuples *x* from table *t* such that $x.a_1 = v_1, \dots$. If v_i is not specified, the value of the a_i attribute of the current tuple is taken. Thus, to delete the tuple corresponding to the displayed content, it is sufficient to specify "**delete T[K1, K2, ...]**" where *K1*, *K2*, ... form a key of *T* and *T* is the base table of this node.

- **update t[a1 = v1, ...]**

updates all the specified tuples from table *t*, set their values for the non key attributes according to what is specified in the **set attr = ...** fields.

keyvalue = keyattrident ["=" first_level_simple_expression]

content = element_type ["(" [field { , field }] ")"]
 | simple_expression
 | "{" field { "," field } "}" .

A content element is either a typed element, made of one or more subfields, or a simple expression. A content element surrounded by { and } is evaluated for each selected tuple. Only one such element may appear in a node definition.

element_type = "<" element_type_identifier { element_attribute_identifier "=" simple_expression } ">"

elem_type_identifier = identifier { (':' | '-') identifier }

element_attribute_identifier = identifier { (':' | '-') identifier }

from_part = ["from"
 table_identifier [alias_identifier] { ',' table_identifier [alias_identifier] }
 ["distinct"]
 select_part group_part order_part
].

group_part = ["group" "by" identifier { "," identifier }]

select_part = ["selected" "by" condition].

order_part = ["order" "by" simple_expression { "," simple_expression }]

condition = ["not"] logical_term { "or" logical_term }

logical_term = logical_factor { "and" logical_factor }

logical_factor = simple_expression (comparison_op simple_expression
 | "is" ["not"] "null")

comparison_op = "like" | "<" | "<=" | ">" | ">=" | "=" | "<>"

simple_expression = ["+" | "-"] term { ("+" | "-") term }

term = factor { ("*" | "/") factor }

```

factor =  stringconstant
        | numberconstant
        | function_identifier "(" simple_expression { "," simple_expression } ")"
        | [table_identifier '.'] attribute_identifier
        | parameter_ident
        | "(" expression ")"

```

Remarks

Aggregate functions (min, max, count, avg) may not appear in content expressions that are repeated for each selected tuple (contents within { and }).

Content expression may not yield a boolean value (because the boolean type does not exist in SQL). This is not forbidden by the grammar, neither checked by the compiler. Boolean contents will generate ill-formed SQL statements.

expression = condition

10 Glossary of terms

Node schema. A text, written in the Lazy language, that defines how to construct a node instance (usually an HTML or XML page). It is comprised of

- a selection part
- a content part (with links)
- and parameters

Node instance. A hypertext node (usually a HTML or XML page) that is part of a hypertext view. It is an instance of a node schema. Its content depends on the schema definition, the actual parameter values, and the current database contents (tables). A node is instantiated each time the user clicks on a link to the node, or when the node is include in another one (see inclusion link).

Node type. The type of content produced when instantiating a node. Currently defined types are : html, xml, pure html (no heading tag is generated), pure xml, svg. The node type is specified at project level, i.e. all the nodes of a project have the same type.

Node server. The role of the node server is to answer to node requests by instantiating the required node schema with the given parameters. It also performs database actions corresponding to active links.

Node compiler. The program that checks the syntax of a node and transforms its different parts into SQL statements (which may still contain placeholders for parameters).

Link. A way to interconnect nodes. The source of a link is a content element of a node, its destination is a node. Lazy links can have four different behaviours: reference (jump), inclusion, expand-in-place, and active .

Reference link. A link from a content element to a node. When the node is instantiated, such a link appears as an undelined text (an HTML anchor). When it is clicked, the referred node is instantiated and replaces the current node (the user "jumps" to the referred node).

Expand-in-place link. A link from a content element to a node. When the node is instantiated, such a link appears as an undelined text (an HTML anchor). When it is clicked, the content of the referred node is instantiated "in-place", it replaces the anchor text.

Inclusion link. A link intended to include the content of the target node withing the source node, at the link location. Inclusion links are useful to create complex node contents.

Active link. A link that triggers a dabase action when it is followed.

Project. A set of nodes that make up a hyperspace intended for a specific purpose. All the nodes of a project have the same type (html, xml, purexml, purehtml) and refer to the same database connection. Every node belongs to exactly one project. Thus the full name of a node has the form projectname.node-name .

Database connection. an access to a particular database (schema). It is comprised of a database URL (e.g. jdbc:hsqldb:hsqldb://localhost), a JDBC driver class (e.g. org.hsqldb.jdbcDriver), a username, and a password. The URL and driver are dbms specific (refer to your dbms documentation).

User. a name under which somebody can connect to a Lazy server. By default the username PUBLIC is used when opening a Lazy session. A user can be defined as "administrator". In this case she will have full access to every node and data.

Datagroup . a symbolic name for a set of data in a database. A tuple in a table may belong to zero or one datagroup, thus the same table can hold data belonging to different applications. For instance, a single database can be used to manage different museums. The datagroup of a tuple is indicated by the value of an attribute reserved for this purpose.

When logging in, a user can select a datagroup, otherwise she will work in her default datagroup.

Role. a role represents a set of access rights to nodes (and database tables). For instance, role VISITOR can acces the node MUSEUM.artist_index while role CURATOR can access MUSEUM.update_catalog, etc. An access right is thus a pair (role, project.node) or (role, table). The wildcard form project.* means "all the nodes of this project". The main advantage of roles is to avoid defining the access rights of each user in each datagroup (see grant).

Grant. when connecting, a user is granted a set of roles. These roles depend on the datagroup she chose to work in. So, for instance, user JOE could have role VISITOR in the MUSEUM_OF_MODERN_ART datagroup and roles VISITOR and CURATOR in CITY_MUSEUM. A grant is thus a triple (user, datagroup, role). If datagroup is * it means that this user has this role in all datagroups.

Servlet.

Servlet container.

11 Bibliography

See <http://cui.unige.ch/isi/reports>

Appendix A. The Lazy dictionary

